IBM3090VF に適した数値計算法(数値計算基本アルゴリズムとそのソフトウェアの研究)

AUTHOR(S):

寒川, 光

CITATION:

寒川, 光. IBM3090VF に適した数値計算法(数値計算基本アルゴリズムとそのソフトウェアの研究). 数理解析研究所講究録 1988, 648: 59-76

ISSUE DATE:

1988-03

URL:

http://hdl.handle.net/2433/100296

数理解析研究所講究録
648巻 1988年 59-76

IBM 3090 VF に適した数値計算法

日本アイ・ビー・エム(株)
寒川 光 (Hikaru Samukawa)

1. はじめに

IBM 3090 ベクトル機構(VF: Vector Facility)は1985年に発表されたベクトル計算機である[1]。3090 VFの命令体系はシステム370 XAを基礎としたものでベクトル・アーキテクチャーと呼ばれる[2]。このアーキテクチャーの特長はシステム370 XAと融和性が高く、31ビット・アドレスの仮想記憶方式(VS方式)を採用している。また、16個のベクトル・レジスターを定義しているが、その長さはアーキテクチャー上はパラメーター(VSS: Vector Section Size)として扱われる。この他、メモリー上のベクトルを汎用レジスターを介して直接使用する(入力として)3オペランド形式のベクトル演算命令や、乗加算や内積のように2つの浮動小数点演算を1つの命令で行う複合演算命令などに特長がある。

以上はアーキテクチャー上の特長であるが、ハードウェア

のインプリメンテーション上の特長を次に記す。VF は3090の各CPUに付加機構として装備可能であり、ベクトル計算に用いられる命令やデータは緩衝記憶制御機構内のキャッシュ・メモリーを経由して供給される。キャッシュ容量は3090のEモデルでは64KBである。ベクトル・レジスター長は128で、ベクトル命令の実行はオーバーラップせず逐次的に行われる。

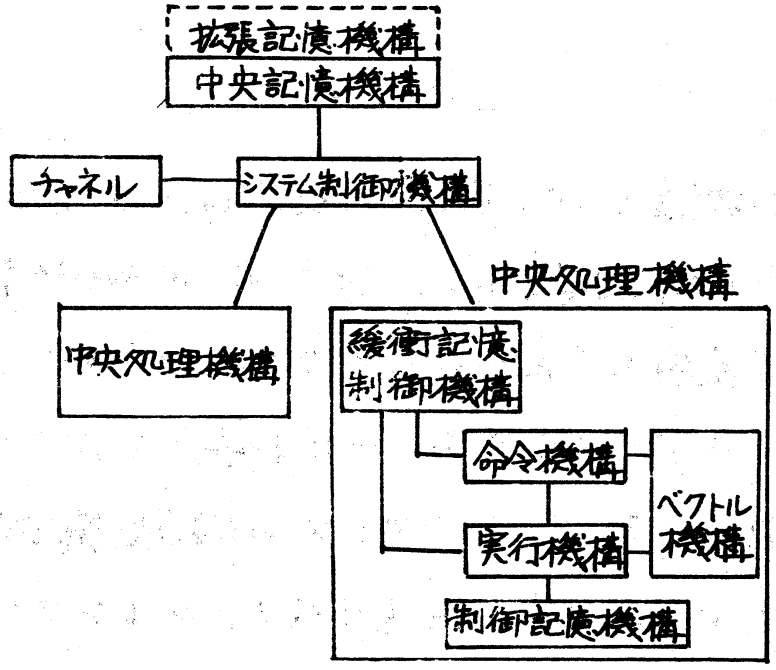


図1. 3090-200E型　構成要素

ベクトル計算機の性能を十分に引きだすためには、ハードウェアの性格にあわせたプログラミングが要求されるが、これを追求するとプログラムの明解さや互換性などのプログラムの一般性を損なう場合が少なくない。将来の計算機はさらに複雑なベクトル命令や並列処理の機能を備え、より個性的になることが予想される。したがって性能とプログラムの一般性という相反する要求を満たすための工夫を考察することは重要である。本稿では線型代数演算を

例に階層型プログラミングを紹介する。この手法は行列とベクトルの積を求めるような単純なサブルーチンをローレベルの階層とし、この部分でハードウェアの個性をできるだけ吸収する方法である。ハイレベルのルーチンはこれらを順次コールする形に作っておけば、システム側の多様化をローレベルのサブルーチンを修正したり入れ換えることで対応可能であり、アプリケーション・アルゴリズムへの影響をくい止めることができる。

2．プログラムの文脈と性能

3090 VFはベクトル・レジスターを備えたVS方式のベクトル計算機であるが、その特長を活かしたプログラミングを紹介する。

2.1　2重ループ

1重ループの例を示す（図2）。ベクトル計算ではループが

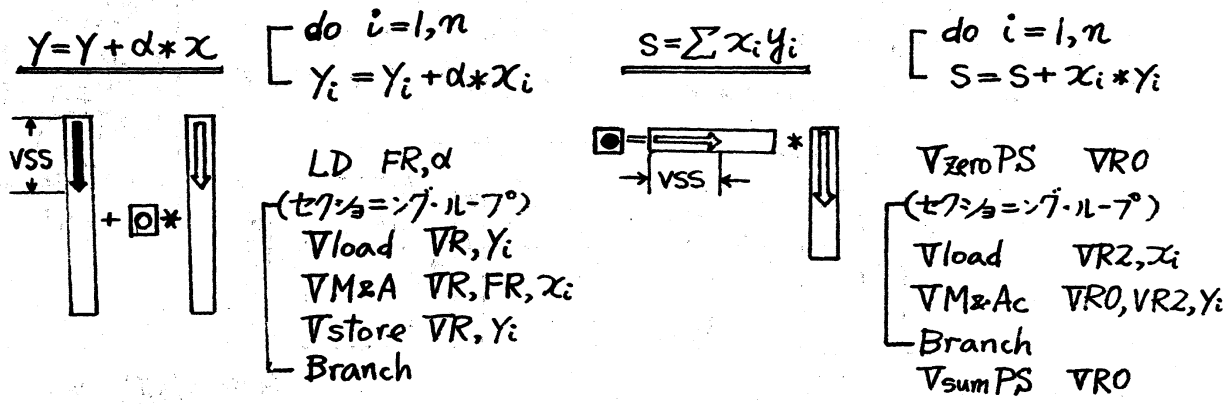


図2．ベクトル・スカラー積 と 内積

ネストすると、ベクトル・ロード／ストア命令を最内側ループから外して性能向上を計ることができる。m行n列の行列とベクトルの乗算を例にあげる。generic form で記した下線部のループ添字に i,j の順でプログラミングするか、j,i の順にするかの2つの方法を考える。図3に両者の処理を示す。この例の添字 j のように右辺に存在して左辺に存在しない添字を本稿ではダミー添字と呼ぶことにする。ベクトル計算でダミー添字が最内側ループの添字に使われるとベクトル演算は内積となり、内側から2番目のループの添字に使われると"2 Vector Operations and 1 Vector-memory Reference"型のベクトル・スカラー積になる。これは図3では VM&A 命令の2つの浮動小数点演算と1回のメモリー参照 (a_{ij}) を指す。

$$Y = Y + A \cdot x$$

```
do __ = 1, __
  do __ = 1, __
    Yi = Yi + aij * xj
```

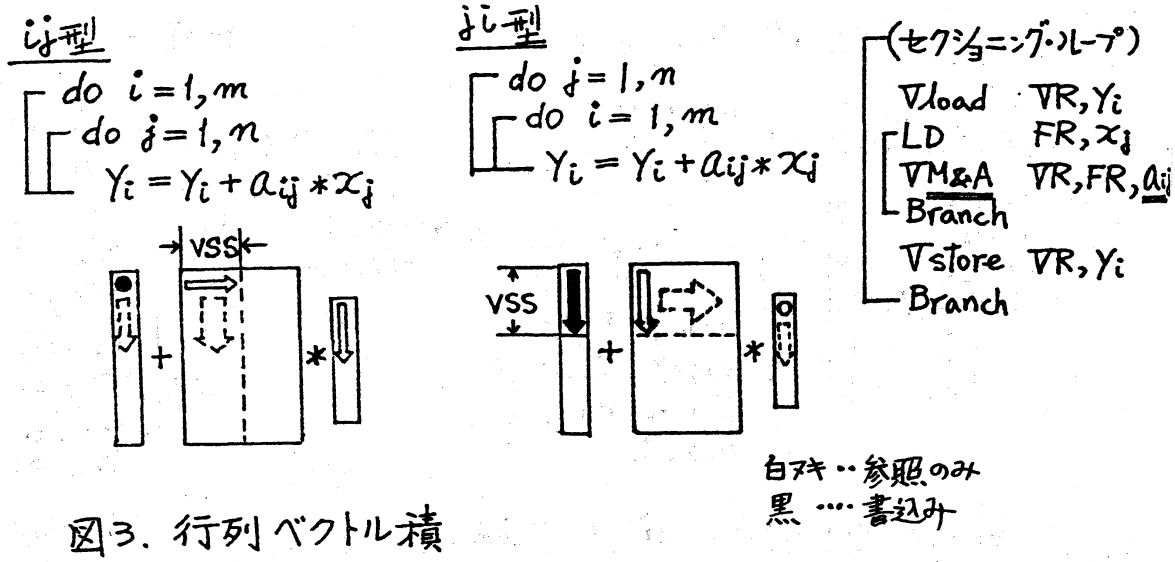


図3. 行列ベクトル積

スカラー計算ではダミー添字を最内側に選ぶと浮動小数点レジスターへ演算結果を足し込めるため、命令数を節約できるが、ベクトル計算では最内側の添字はベクトル化されるため、内側から2番目に配置することによってベクトル・レジスターに足し込みが行える。階層型プログラムを考えるとき、ベクトル計算とスカラー計算の違いはここに現われる。すなわちスカラー計算では2重ループ演算は1重ループ・サブルーチンをコールするかたちに構成しても大きな性能差は生じないが、ベクトル計算で同じ構成をとると"2 Vect.Op. 1 Vect.Ref"による性能を享受できない。このため2重ループ・サブルーチンを単独のサブルーチンとして用意する必要がある。

2.2 3重ループ

行列の積 $C_{ij} = C_{ij} + \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$ をgeneric formで記す。添字 i, j, k に関するループ選択は $3! = 6$ 通り存在する（図4）。この中では行列ベクトル積と同じ"2 Vect.Op. 1 Vect.Ref."型で、かつメモリー上のベクトルを連続的に参照するjki型の性能が最も高い。

```
do __=1,__
  do __=1,__
    do __=1,__
      Cij = Cij + aik * bkj
```

(A: l×m, B: m×n, C: l×n)

IBM3090のようにキャッシュを備えた計算機では、メモリー参照されるベクトル・データ（a_{ik}）がキャッシュ内に存在し

た時の方が（メモリーからキャッシュ経由で参照される時よりも）命令実行時間が短い。３重ループの文脈ではこの特質をも考慮して、２重ループの性能（MFLOPS性能値比較）を測ることができる。具体的には、jki型のkのループ回数を、演算に必要なデータがキャッシュに収まりきる範囲でいったん中断し、先にiを進める計算順序をとる（図5）。

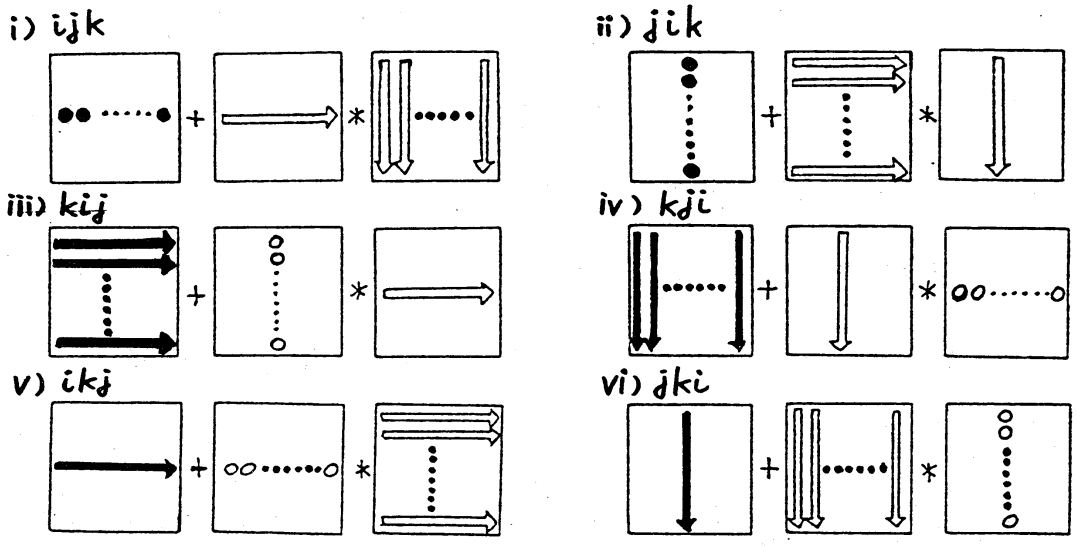


図4. 行列積の6つのループ選択

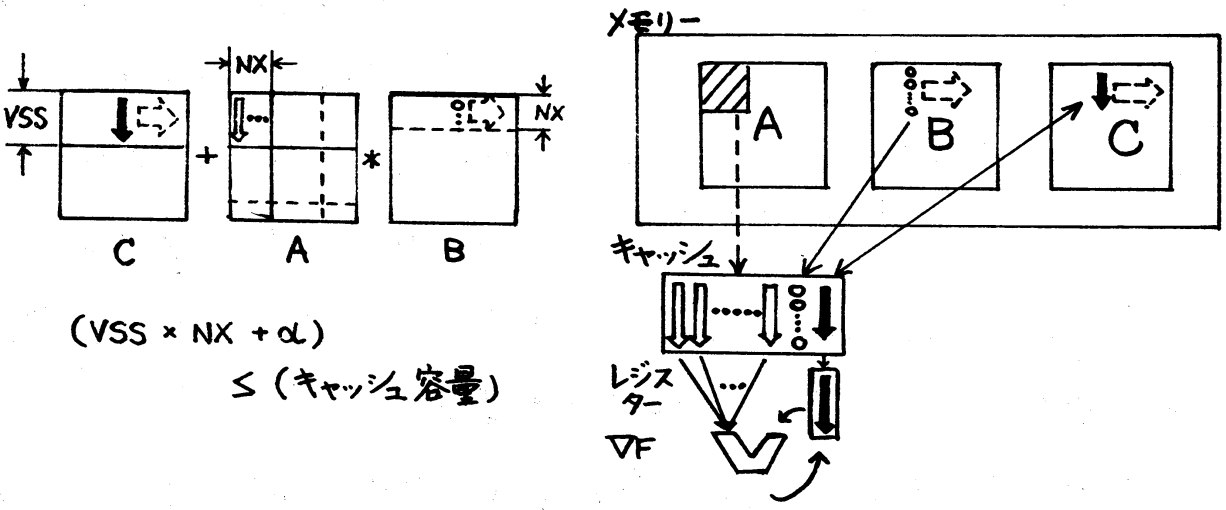


図5. キャッシュ容量を意識した行列積

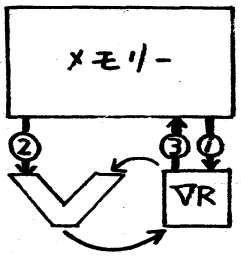
2.3 メモリー階層と階層型プログラミング

階層型メモリー構造をもったベクトル計算機ではループのネストが深くなるに従って計算性能を高めることができる。

・1重ループ

```
do i=1,n
  Yi = Yi + α*xi
```

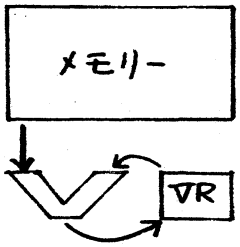
```
∇load  ····①
∇M&A   ····②
∇store ····③
```



・2重ループ

```
do
  do i=1,n
    Yi = Yi + xj * aij
```

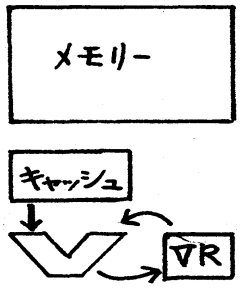
```
∇M&A  ∇R, FR, aij
```



・3重ループ

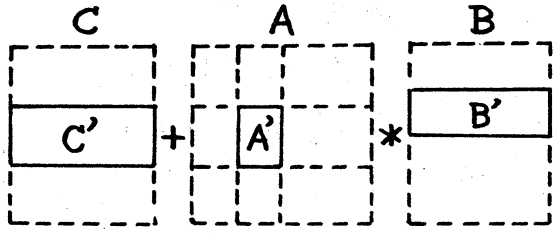
```
do
  do
    do i=1,n
      Cij = Cij + bkj * aik
```

```
∇M&A  ∇R, FR, aik
                in cache
```



最内側ループがFORTRANステートメントとしては同じベクトルスカラー積を表現していても、1重ループでは3つのベクトル命令が用いられ、2重ループでは1つに減らせる。3重ループではオペランドをキャッシュにキープすることが可能になる。こうしたシステム側の特性を数値計算アルゴリズムの表面にあらわにすることは、プログラムの一般性の観点からは好ましくない。行列積を例に、システム側の特性を覆い隠したプログラムの例を示す。

```
Subroutine DMPYAD (A,LDA,B,LDB,C,LDC,L,M,N)
double precision A(LDA,M),B(LDB,N),C(LDC,N)
common /SYSTEM/ LVS,NCACHE
                              〔… C=C+A*B〕
NZ = MIN (LVS, L)
NX = NCACHE / NZ
NII = (L-1) / NZ + 1
NJJ = (M-1) / NX + 1
 do  ii = 1, NII
   i = (ii-1)*NZ + 1
   NZL = MIN(L-i+1, NZ)
  do jj = 1, NJJ
     j = (jj-1)*NX + 1
     NXM = MIN (M-j+1, NX)
     call DMULAD ( A(i,j), LDA, B(j,1), LDB,
                   C(i,1), LDC, NZL, NXM, N )
  enddo        〔… C'=C'+A'*B'〕
 enddo
 return
```



COMMON/SYSTEM/ にベクトル・レジスター長（LVS）と適切なキャッシュ容量（NCACHE）などのシステム依存パラメーターを持つ。また、サブルーチン DMULAD は $C'=C'+A'*B'$ を計算するが、行列 A' のサイズは NZL × NXM であり、NZL はベクトル・レジスター長よりも小さいため、セクショニング・ループを無視した2重ループ・サブルーチンである。[注] ループ選択順序やキャッシュ容量を意識した計算順序は図5に示したようにかなり複雑なものであるが、プログラムを階層化する

ことによって小行列の積和という単純なアルゴリズムにまとめることができる。また、ベクトル・レジスター長やキャッシュ容量の異なる機種に対しても、SYSTEMコモンの初期化で対応することができるし、スカラー計算の環境では、ローレベル・サブルーチンDMULADをスカラー用にチューニングしたものに置き換えることで対応することができる。（キャッシュ容量を意識することは、スカラー計算の環境下でも性能向上につながる。）このようにプログラムをよりプリミティブな機能のサブルーチンに分解し、アーキテクチャーやハードウェアに依存する差異をローレベル・サブルーチンのなかで解決してしまうことが、性能とプログラムの一般性の両立のためには実用的な方法である。より複雑な演算では、さらにこの行列積を使用するようにプログラムを構築してゆく。

【注】

最もローレベルのサブルーチンDMULADのループ部分の概略は右のように構成される。キャッシュ容量を意識したこの方法と通常のjki型の性能差は、3090のEモデルでは各行列を次数1000の正方行列とした場合約20%程度である。

```
┌─ Vload    VR, Cij
│ ┌ LD      FR, Bkj
│ │ VM&A    VR, FR, Aik
│ └ Branch  ····(k=1,NXM)
│ Vstore    VR, Cij
└ Branch    ····(j=1,N)
```

（ベクトル命令は i=1,NZL(≦VSS) について行う）

3．連立1次方程式の直接解法

密行列を係数行列とした連立1次方程式を三角分解（LU分解）によって解く問題を考える。行列積の問題と同じように、まずループの構成を考え、次にキャッシュ容量を意識した方法を考える。

三角分解は通常もとの正方行列Aを2つの三角行列の積に変換する。

$$A = LU$$

行列Lは対角項を1にした下三角行列、Uは上三角行列である。一般にLとUを計算するアルゴリズムは始めにAが書かれていた領域に書き込まれてゆく。すなわち、配列Aに正方行列を与えられると、配列Aの上三角行列の部分にUを、下三角行列の対角項を除いた部分にLの対角項よりも下の部分を生成する。

3.1 ループの選択（6通りのループ選択）

三角分解のアルゴリズムを generic form で記す[3]。ただし、枢軸（ピボット）選択はここでは省略した。generic form は

```
do __= __,__
  do __= __,__
    do __= __,__
      a(i,j) = a(i,j) - (a(i,k) * a(k,j)) / a(k,k)
```

（添字の1例）
k= 1, n-1
j= k+1, n
i= k+1, n

行列積に類似しているが、ループ添字の動く範囲が外側のループ添字で記されるため、はるかに複雑である。この3重ループに対する6通りの選択を図6に示す。ダミー添字kをまん中に選んでもベクトル・レジスターに足し込みを行う。"2 Vect.Op. 1 Vect.Ref."型の命令は生成されない。これは更新されるベクトルが2重ループの文脈のなかで変化するからである（最内側ループの添字i/jの動く範囲がひとつ外側のループ添字kで記述されている）。一般にこのような文脈にループ・アンローリングを施こして、ベクトル演算に対するメモリー参照の回数

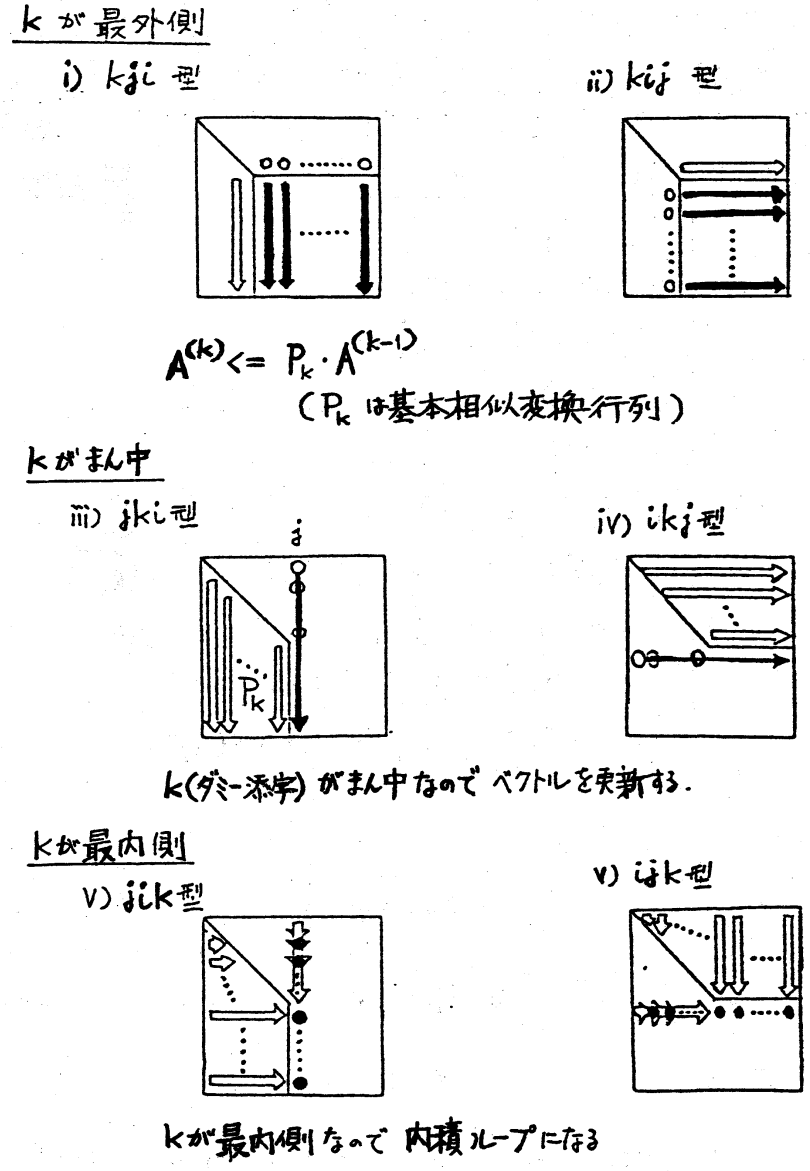


図6. 三角分解のループ選択

を減らすことができる。

3.2 ループの選択（6通り以外の方法）

6通りのループ選択のなかで、jki型の後半（下三角行列の要素を計算する部分）とijk型の後半（上三角行列の要素を計算する部分）は、"2 Vect. Op. 1 Vect. Ref." 型でメモリー参照も連続的になっている。両者の利点を合わせてループを構成してみる。図7にこの方法（ijk·jki型）の処理を図示

```
do ij = 1, n
  i = ij
  do j = i, n
    do k = 1, i-1
      a(i,j) = a(i,j) - a(i,k) * a(k,j)
  j = ij
  do i = j+1, n
    a(i,j) = a(i,j) / a(j,j)
  do k = 1, j-1
    do i = j+1, n
      a(i,j) = a(i,j) - a(i,k) * a(k,j)
```

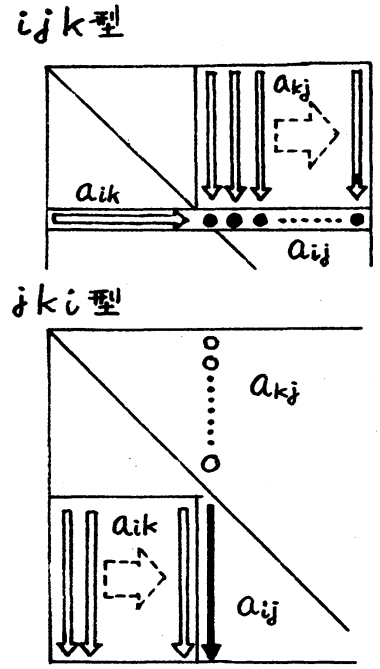


図7. ijk·jki型

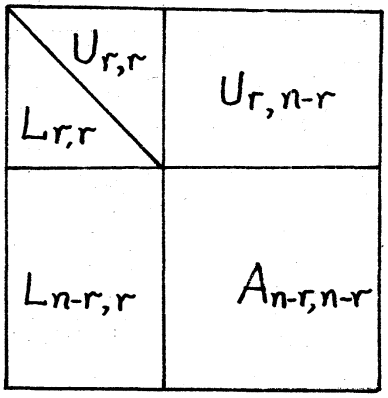
した。ここでは最内側ループの添字の動く範囲は2つ外側のループ添字で記述されているため、長方形行列を対象に計算を進めることができる。

ijk·jki型はループ・アンローリングを施こしたjki型に匹敵した性能を示す。しかも内側の2重ループは「連続する内

積」（あるいは「転置行列とベクトルの積差」）と「行列とベクトルの積差」という単純なアルゴリズムのサブルーチンにより処理することができるので、プログラムの一般性を保ちやすい。これはループ・アンローリング法がプログラムを冗長かつ複雑にすることと比べると大きな利点といえる。

3.3 キャッシュ容量の考慮

ijk・jki型の処理をrステップ実行した状態では、配列Aの右下$(n-r)\times(n-r)$の小行列は最初の状態のままで残っており、他の小行列は最終結果の三角行列の小行列になっている（右図）。



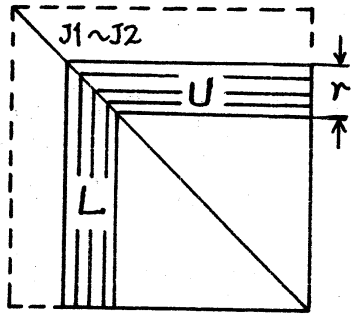
ここで行列の積差

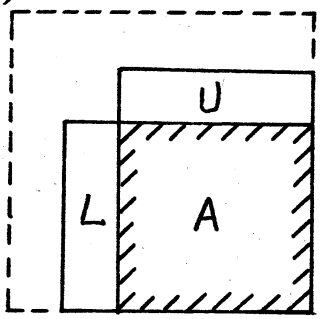
$$A'_{n-r,n-r} = A_{n-r,n-r} - L_{n-r,r} * U_{r,n-r}$$

を計算すると、三角分解の問題を次数$n-r$の問題に縮小することができる。[4] この処理を繰返して順次問題を縮小し、最終的な三角行列LとUを求めることができる。(各ブロック・ステップの前半はijk・jki型のプログラムを一部修正して利用することができ、また後半では行列積のサブルーチンを利用することができる。)

$$A'_{n-r,n-r} = L_{n-r,n-r}\ U_{n-r,n-r}$$

```
Subroutine DIJKIC (A, LDA, N, IPVT)
double precision A(LDA,N)                    [A => L·U]
integer IPVT(N)
common /SYSTEM/ ....
    :
MR = r
 do  M = 1, MSTEP
   J1 = (M-1)*MR + 1
   J2 = MIN (N, J1+MR-1)
    call DIJKIT (A , LDA, N, IPVT, J1, J2)
                [J1~J2 行を三角分解する]
    call DMPYSB ( A(J2+1 , J1) , LDA , A(J1 , J2+1), LDA
                , A(J2+1, J2+1), LDA, N-J2, J2-J1+1
                , N-J2 )
                [A = A - L·U  を計算する]
 enddo
return
```





(DMPYSB は 2.3節の DMPYAD に対応するサブルーチン)

図8. キャッシュ容量を意識した三角分解

この方法は行列Aをあらかじめ複数の小行列に分割してファイル上に置き，入出力を伴ないつつ消去を行う分割消去法（ブロック消去法）に類似する。しかしここでは，行列Aをメモリー（仮想記憶域）に置いているため，ファイル上でのフォーマットに制約されない自由なブロッキング・サイズをとることができる。事実，各ブロック・ステップの後半で行なう行列の積差演算（DMPYSB）では，他の部分とは独立に小行列のサイズを選択することができる。

現実のプログラミングでは，性能に大きな影響を与えるためハードウェアを意識せざるをえないルーチンを machine dependent routines としてあらかじめインターフェースを定義し，アルゴリズムの表面にまでこうした要素（ベクトル計算かスカラー計算，ベクトル・レジスター長，キャッシュ容量，プロセッサーの数など）が顔を出すのを抑えるべきである。3090 VF は VS方式のベクトル計算機であるため，データ（行列 A）を単一のアドレス空間に保持できる。したがって，階層化されたプログラムによって性能とプログラムの一般性の両立を実現しやすい計算機である。

3.4 性能の比較

図9 に相対的な性能比を示す。行列の次数 1000 までについて，jki型，ijk-jki型，DIJKIC（キャッシュ容量を考慮

した方法）の3者を比較した。jki型とijk-jki型の差はループ構成の差（"2 Vect. Op. 1 Vect. Ref."型と"2 Vect. Op. 3 Vect. Ref"型の差）によるものである。DIJKICはキャッシュ容量を考慮した演算部分（行列の積差を計算する部分）の比率が、行列の次数が大きくなるに従って高くなるため、ijk-jki型との差が開いてくる。

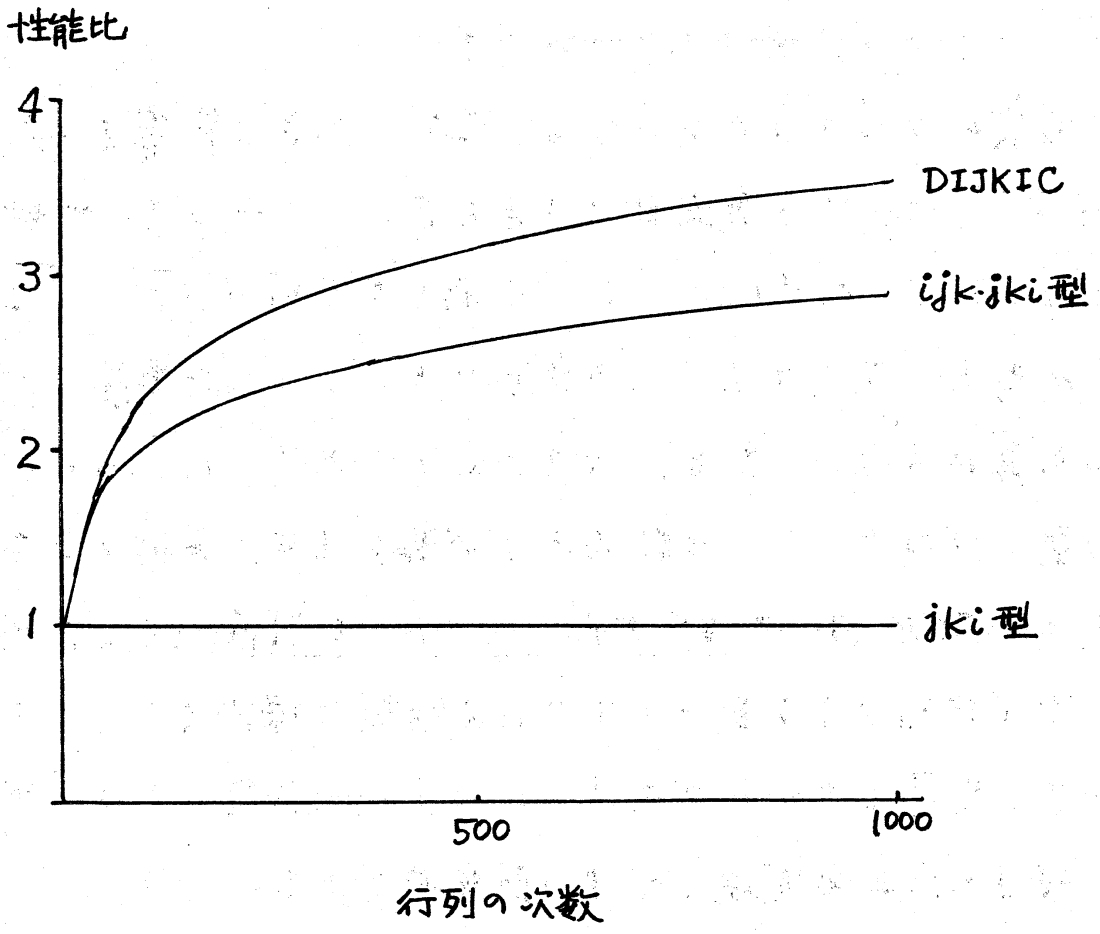


図9. 相対性能比

4. おわりに

技術計算の計算能力に対する需要増大をサポートするために、ハードウェアのからくりはベクトル計算、並列計算をとりこんでますます複雑化してゆく傾向にある。FORTRAN はアセンブラーを透視して性能を意識したコーディングをする余地を残した言語である。また、サブルーチン・コールやコモンの利用による融通性も備えている。本稿ではこれらの特質をうまく使うことにより、アルゴリズムのエレガンスを保ちつつ性能を追求し、さらに将来のハードウェアのインプリメンテーション変化にもわずかな調整で対応する方法を考えてみた。この手法の実現には、システム（アーキテクチャー，ハードウェア）に依存する部分を単純な機能に絞り込み将来の変化による影響をこの部分におしとどめることが、実用的な方法のひとつである。

〔参考文献〕

1. Gibson, D.H., Rain, D.W. and Walsh, H.F., "Engineering and scientific processing on the IBM 3090", IBM Systems Journal, Vol.25, No.1, 1986
2. "IBM System/370 Vector Operations", SA22-7125, 1986

3. Dongarra, J.J., Gustavson, F.G. and Karp, A., "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine", SIAM Review, Vol. 26, No. 1, Jan., 1984
4. Wilkinson, J.H., "The Algebraic Eigenvalue Problem", Clarendon Press, Oxford, 1965